Grafika Komputerowa i Komunikacja Człowiek-Komputer

Data zajęć: 26.11.2017 Data oddania: 24.01.2018
Termin zajęć: Pon. TP 9:15 Prowadzący zajęcia: Mgr inż. Szymon Datko

Sprawozdanie nr 4

Jakub Majewski 238902

---

1. Opis tematu

Temat: OpenGL - Oświetlenie scen 3-D.

Zrealizowane zadania:
1. stworzenie pojedynczego źródła światła przy użyciu biblioteki OpenGL,
2. stworzenie tablicy wektorów normalnych dla własnego modelu,
3. dodanie drugiego źródła światła o innej barwie.

2. Opis najważniejszych fragmentów kodu

1. Metody klasy 'Light' odpowiedzialnej za tworzenie pojedynczego źródła światła.

```
Light::Light(int idx ) : idx(idx), glIdx(glIdx),
  light_position{ 0.0, 0.0, 10.0, 1.0 },
  light_ambient{ 0.0, 0.0, 0.0, 1.0 },
  light_diffuse{ 1.0, 1.0, 1.0, 1.0 },
  light_specular{ 1.0, 1.0, 1.0, 1.0 },
  att_constant{ 0.f }, att_linear{ 0.f },
  att_quadratic{ 0.f } {;}

// Inicjalizacja światła z domyślnymi parametrami
void Light::Init() {
  glLightfv(glIdx, GL_AMBIENT, light_ambient);
  glLightfv(glIdx, GL_DIFFUSE, light_diffuse);
  glLightfv(glIdx, GL_SPECULAR, light_specular);
  glLightfv(glIdx, GL_POSITION, light_position);
  glLightf(glIdx, GL_CONSTANT_ATTENUATION, att_constant);
  glLightf(glIdx, GL_LINEAR_ATTENUATION, att_linear);
  glLightf(glIdx, GL_QUADRATIC_ATTENUATION, att_quadratic);
}

Light& Light::SetCLQ(float c, float l, float q) {
  att_constant = c; att_linear = l; att_quadratic = q;
  glLightf(glIdx, GL_CONSTANT_ATTENUATION, att_constant);
  glLightf(glIdx, GL_LINEAR_ATTENUATION, att_linear);
  glLightf(glIdx, GL_QUADRATIC_ATTENUATION, att_quadratic);
  return *this;
}

Light& Light::SetPosition(float x, float y, float z) {
  light_position[0] = x;
  light_position[1] = y;
  light_position[2] = z;
  glLightfv(glIdx, GL_POSITION, light_position);
  return *this;
}

// Analogicznie jak dla SetAmbient:
Light& Light::SetAmbient(float x, float y, float z) {...}
Light& Light::SetDiffuse(float x, float y, float z) {...}
Light& Light::SetSpecular(float x, float y, float z) {...}
```

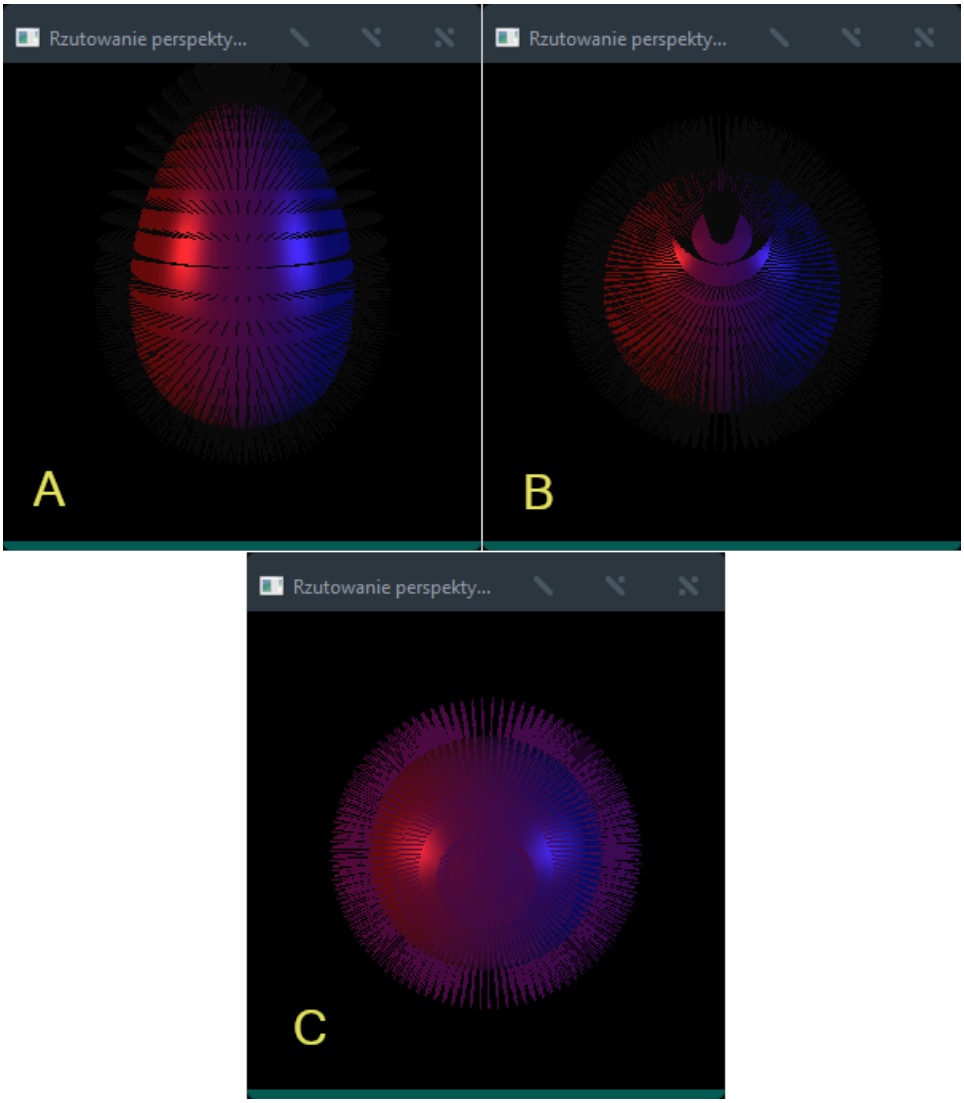
2. Klasa 'LightManager' odpowiedzialna za przechowywanie i tworzenie nowych źródeł światła. Umożliwia również modyfikowanie parametrów już stworzonych świateł.

```
struct LightManager {

  GLfloat mat_ambient[4];
  GLfloat mat_diffuse[4];
  GLfloat mat_specular[4];
  GLfloat mat_shininess;

  std::vector<std::unique_ptr<Light>> lights;

  LightManager() :
    mat_ambient{ 0.2, 0.2, 0.2, 1.0 },
    mat_diffuse{ 1.0, 1.0, 1.0, 1.0 },
    mat_specular{ 1.0, 1.0, 1.0, 1.0 },
    mat_shininess{ 100.0 } {;}

    // Inicjalizacja silnika świateł OpenGL oraz parametrów
    ↪  powierzchni
  void Init() {

    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_LIGHTING);
    glEnable(GL_DEPTH_TEST);
  }

    // Zwracanie istniejącego już światła
  Light* GetLight(int idx) {
    return lights[idx].get();
  }

    // Tworzenie nowego światła
  Light* CreateLight() {
    glEnable(GL_LIGHT0+lights.size());
    lights.emplace_back(new Light(lights.size()))->Init();
    return lights.back().get();
  }
};
```

2. Metody w odpowiedzialne za wygenerowanie przestrzeni wektorów normalnych renderowanego obiektu.

```
1  void Egg::calculateTU(float *tu, float u, float v) {
2    tu[0] = (-450.f * u*u*u*u + 900.f * u*u*u - 810.f * u*u +
     ↪  360.f * u - 45.f) * std::cos(3.14159f*v);
3    tu[1] = (640.f * u*u*u - 960.f * u*u + 320.f * u);
4    tu[2] = (-450.f * u*u*u*u + 900.f * u*u*u - 810.f * u*u +
     ↪  360.f * u - 45.f) * std::sin(3.14159f*v);
5  }
6
7  void Egg::calculateTV(float *tv, float u, float v) {
8    tv[0] = 3.14159*(90.f * u*u*u*u*u - 225.f * u*u*u*u + 270.f *
     ↪  u*u*u - 180.f * u*u + 45.f * u) * std::sin(3.14159f*v);
9    tv[1] = 0.f;
10   tv[2] = -3.14159*(90.f * u*u*u*u*u - 225.f * u*u*u*u + 270.f
     ↪  * u*u*u - 180.f * u*u + 45.f * u) * std::cos(3.14159f*v);
11 }
12
13 void Egg::calculateNormalTab() {
14   float tu[3]{}, tv[3]{};
15   for (int i = 0; i < N; ++i) {
16     for (int j = 0; j < N; ++j) {
17       float u = float(i) / (N - 1);
18       float v = float(j) / (N - 1);
19       calculateTU(tu, u, v);
20       calculateTV(tv, u, v);
21       float *p = normalTab[i][j];
22       p[0] = tu[1] * tv[2] - tu[2] * tv[1];
23       p[1] = tu[2] * tv[0] - tu[0] * tv[2];
24       p[2] = tu[0] * tv[1] - tu[1] * tv[0];
25       float len = std::sqrt(p[0]*p[0]+p[1]*p[1]+ p[2]*p[2]);
26
27       // Kod naprawiający, opisany we wnioskach
28       if(i == 0 || i == N-1) p[1] = -1.f;
29       else {
30         if (u >= 0.5f) len *= -1.f;
31         p[0] /= len;
32         p[1] /= len;
33         p[2] /= len;
34       }
35     }
36   }
37 }
```

3. Rezultat pracy



Rysunek 1: Obiekt oświetlony dwoma źródłami światła (o różnej barwie), obserwowany z boku (A), od góry (B) oraz od spodu (C)

4. Spostrzeżenia i wnioski

Po stworzeniu przestrzeni wektorów normalnych promienie świetlne zaczęły się odbijać od obiektu w naturalny i prawidłowy sposób, ale na spodzie obiekt dalej nie był oświetlony. Wynikało to z faktu, że długość obliczonych wektorów normalnych na spodzie jajka była równa zero. W wyniku normalizowania wektorów (dzielenia ich parametrów przez długość wektora), parametry wektora osiągały wartości typu Nan. Problem rozwiązałem przez stworzenie w newralgicznych punktach sztucznych wektorów o parametrach [0, -1, 0].